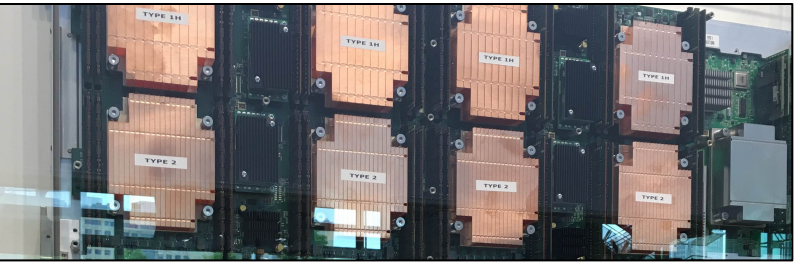
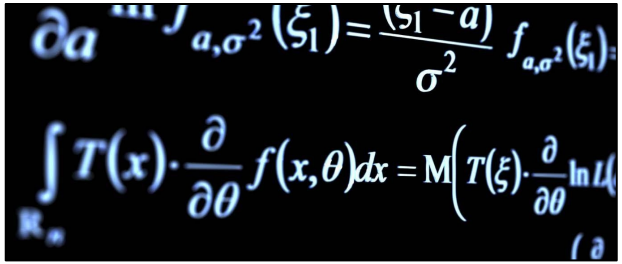
Exceptional service in the national interest

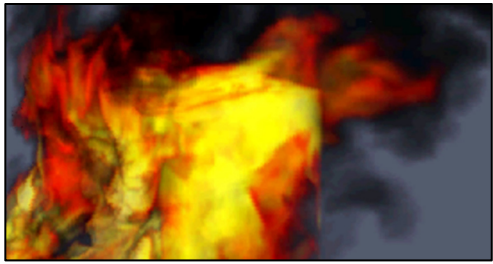
Sandia National Laboratories





$$\frac{\partial}{\partial a} f_{a,\sigma^2}(\xi_1) = \frac{(\xi_1 - a)}{\sigma^2} f_{a,\sigma^2}(\xi_1)$$

$$\int T(x) \cdot \frac{\partial}{\partial \theta} f(x,\theta) dx = \mathrm{M}\left(T(\xi) \cdot \frac{\partial}{\partial \theta} \ln L \right)$$



On the Importance of Faster Atomics

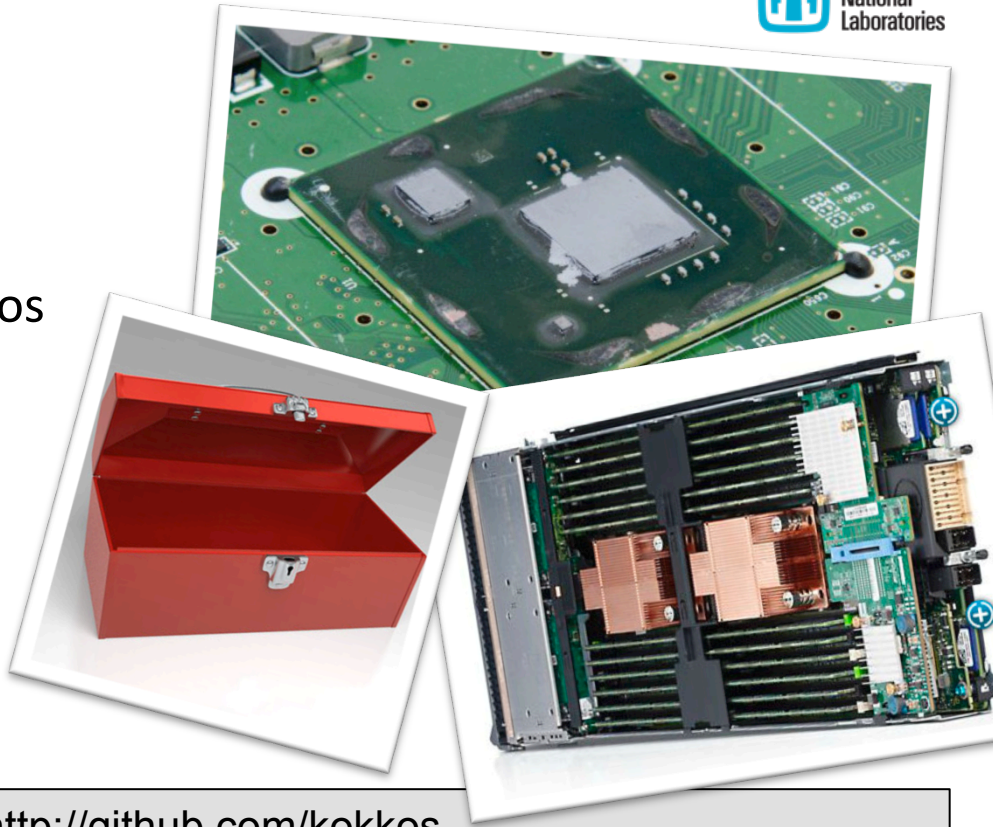
S.D. Hammond, C.R. Trott and H.C. Edwards, Center for Scientific Computing
Sandia National Laboratories/NM

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

- Motivations and Background
- Exposing Atomic Operations in Kokkos
- Performance
- Conclusions



More Information: http://github.com/kokkos

Motivations

Sandia National Laboratories

- **Sandia is heavily focused on making sure that our production application codes will run well on current and future NNSA Advanced Technology System (ATS)**
  - ATS-1 – Trinity (~9,500 dual-socket Haswell, ~9,500 single-socket KNL)
  - ATS-2 – Sierra (~4,000 POWER9/Volta (2018))
  - ATS-3 – Crossroads ? (2020)
- **For all of these platforms we need to have performance portable algorithms and source code**
  - Kokkos for C++ Applications
  - OpenMP for Fortran

Motivations

Sandia National Laboratories

- **Enabling performance portable, on-node parallel algorithms can be extremely challenging:**
  - Correctness (developer dependent, some tools to help)
  - Portability (Kokkos helps, but developer work still required)
  - Performance (heavily developer dependent)
- **In order to meet our objectives to have applications running on these machines as quickly as possible**
  - Need to keep changes to code to a relative minimum
  - Keep initial algorithms similar to prevent significant re-development/re-coding efforts

Atomic Operations

Sandia National Laboratories

- **Atomic operations in many ways are an application enabler:**
  - Keep roughly serial algorithms but provide atomic updates to (limited) regions of memory which threads may share
  - Keep code changes to a relatively minimum
  - Isolate expensive memory updates to where they *need* to be
- **Disadvantages in applications:**
  - Floating point rounding differences (floating point ops are not associative)
  - Variation in runtimes if contention rates/effects change between runs
  - Can be expensive
- **Required for lock-free shared data structures**
  - Queues, hash-maps, ...

Alternatives to use of Atomic Operations

Sandia National Laboratories

- **Requires new algorithms (e.g. coloring/data replication) to be implemented:**
  - Expensive in application developer time
  - Don't always have enough parallelism to support coloring schemes
  - Significant code churn
  - Consumes vast amount of memory if thread count high (data replication)
- **Advantages of alternatives are:**
  - Potentially higher performance (if we have enough parallelism)
  - Less performance variation between runs because very little shared resources
  - Strong reproducibility of results

Exposing Atomics in C++

Sandia National Laboratories

- C++11 introduced atomic memory updates into the standard
- But … std::atomic is fairly clunky, requires specific allocations *etc*.

```
std::atomic<int> data;

void updateMe() {
   data.fetch_add(1, std::memory_order_relaxed);
}
```

- We really want something simpler and easier to use
  - A fix has been *proposed* for C++20

Exposing Atomics in Kokkos

Sandia National Laboratories

- Don't require "atomic" types (operate over any type, including non-POD)
- Implement a lightweight locking system based on pointer address for types not supported by hardware atomics/CAS

```
int data;

void updateMe() {
   Kokkos::atomic_fetch_add(&data, 1);
}
```

- Much simpler to use, can atomically update *any* value and does not propagate through the type system

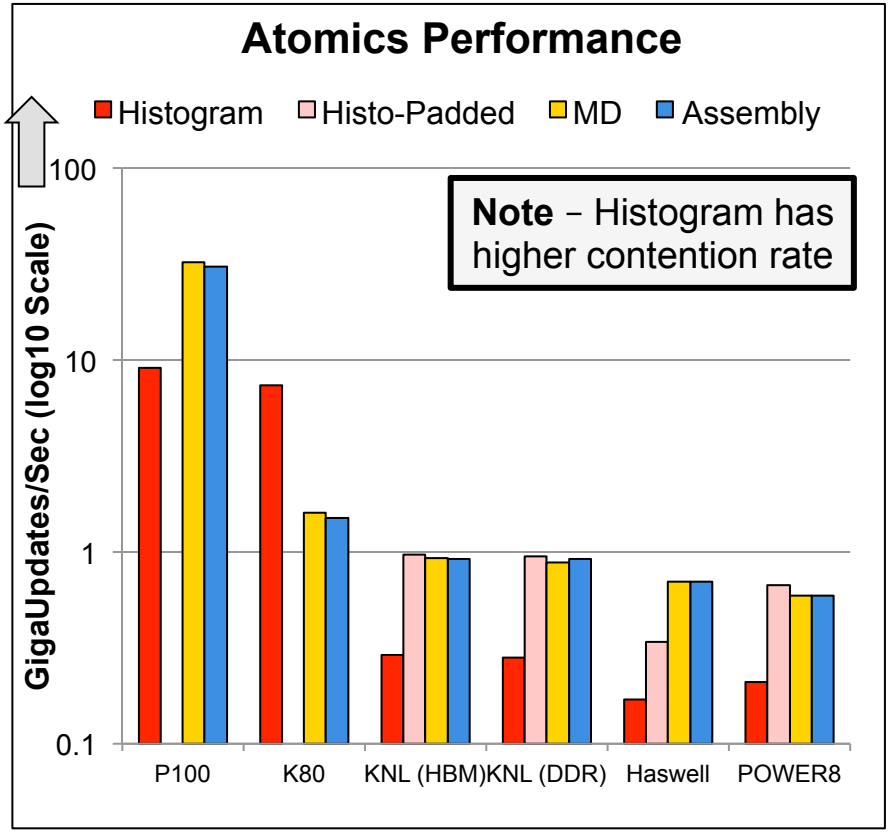
Performance of Atomic Operations

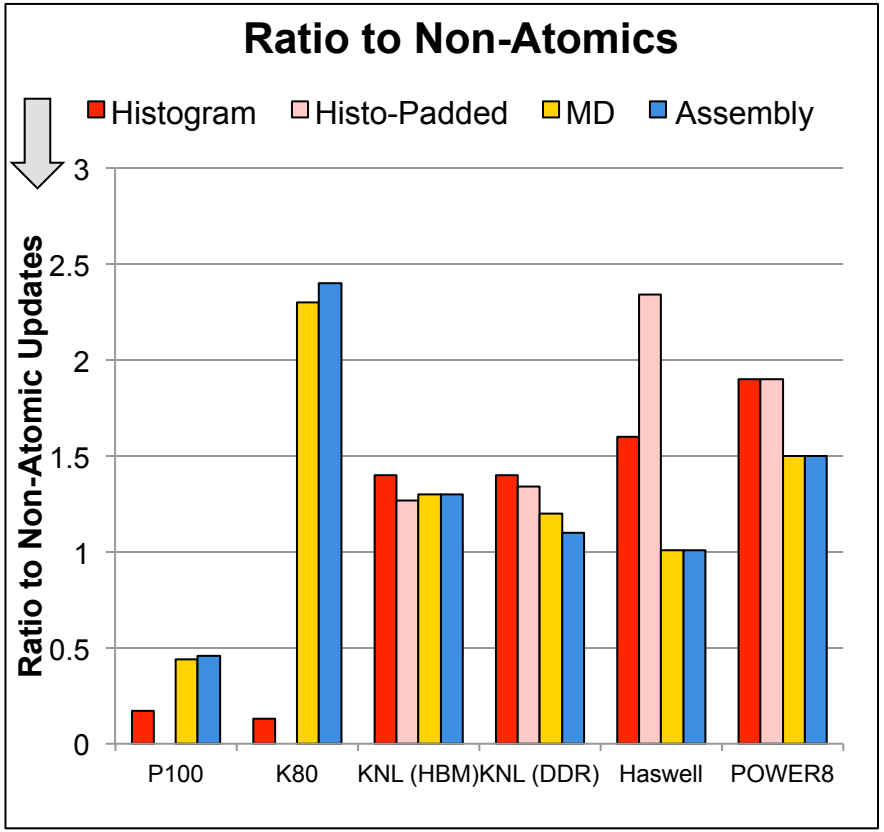
Sandia National Laboratories

- **We have developed three rough "categories" of atomic-issue rate and contention levels from some of our initial application ports:**
  - Histogram (count values in a bin in parallel and update, integers)
  - MD (LAMMPS like use of atomic updates to reduce duplicate work,double)
  - Matrix Assembly (accumulate values into a matrix from an unstructured mesh, double)
- **Run on our current systems:**
  - GigaUpdates per second
  - Ratio of using atomics to standard memory operations (i.e. atomic overhead)
  - Run in the "best configuration" (Fastest use of OpenMP/processes, Single Socket for CPU systems)
  - Ratio to non-atomic is performance against not using atomics (incorrect answers)

Performance of Atomic Operations

Sandia National Laboratories





Histo-Padded provides padding for cache lines to prevent conflicts (uses more memory)

Discussion

Sandia National Laboratories

- Atomics are clearly very fast on the latest generation of NVIDIA Pascal (P100) GPUs due to hardware enablement at the cache (“fire and forget”)

- CPUs and historically struggled with fast atomic updates because they add a significant number of additional operations into the instruction stream
  - and .. Cache line sharing, inability of compiler to easily optimize around

- **Faster atomics** on these platforms *and* easier ways to program atomics would make **algorithm development for next-generation platforms easier**, **reduce programmer burden** and **improve compiler information for analysis**

Discussion

Sandia National Laboratories

- Most algorithms have relatively low (but non-zero) contention rates
  - Atomics are really used to enable correctness for the very limited cases there is a shared data conflict
  - But ... the overhead is high for the operations where no contention occurs

Conclusions and Position

Sandia National Laboratories

- **Atomic Memory Operations** are potentially a lightweight programming choice to introduce thread safety and parallelism to existing code
  - Use atomics to update memory locations you know may have conflicts
- C++11 introduced atomics to the language standard but the method of use is less than ideal for minimizing code changes
  - Fix has been *proposed* for C++20
- Kokkos provides a lightweight, use anywhere implementation for C++ codes
- **Need better hardware support to reduce the overheads in our applications**

More Information: http://github.com/kokkos


Sandia
National
Laboratories